RDMA User←→Kernel kABI

Status and going forward

Matan Barak

Session Goals

- Making the new kABI enabled by default
- Pending Tasks
- Community Effort

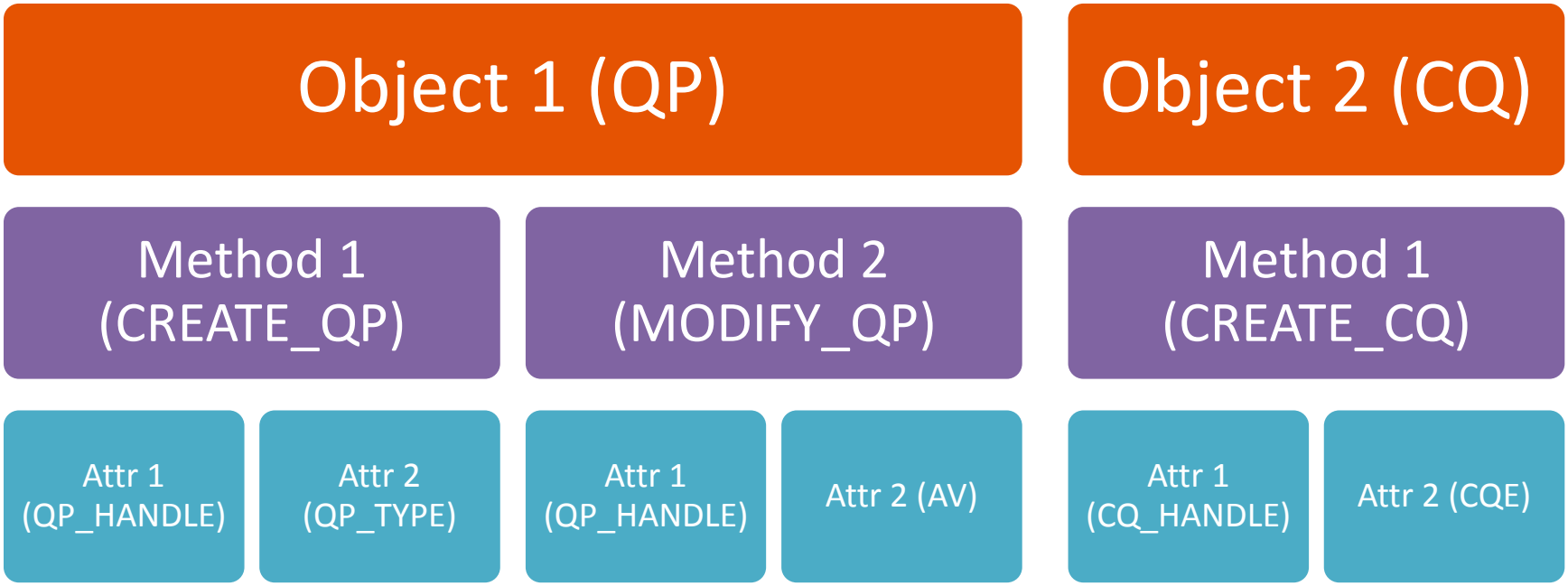
BoF

Introducing the kABI core concepts VERY briefly here, for more information: patches, presentation, me ☺

kABI Goals

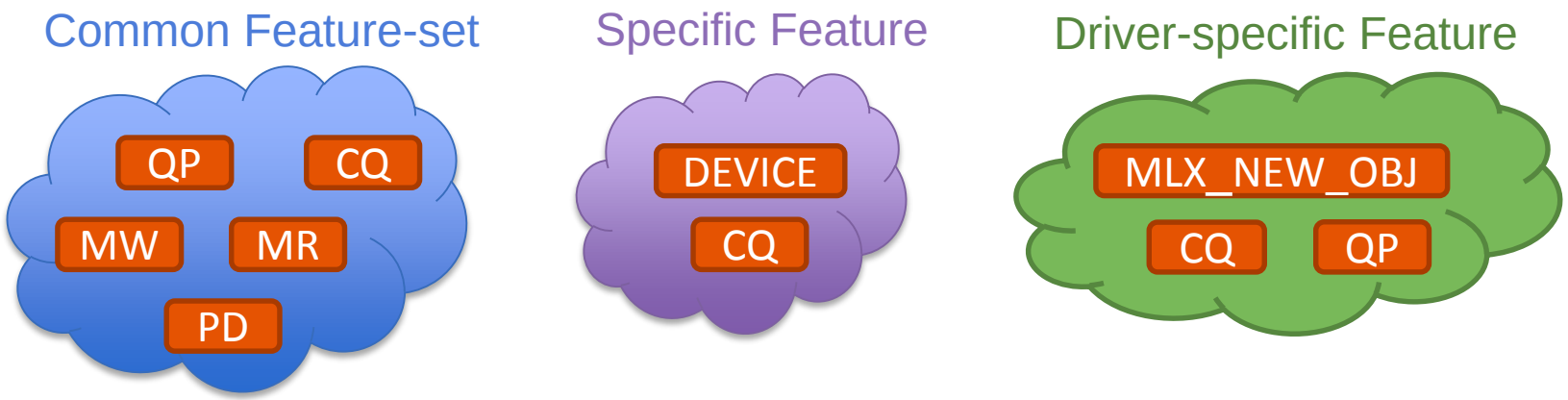
- Resolving write() security issue
- Introducing a well defined extensible approach
  - Existing methods (“verbs”) are easily extensible
  - Drivers could add their special sauce
    - New objects, methods and attributes
    - Mix driver specific methods and attributes for existing objects and methods.
- Future syntactic based capability system
  - Grouping all aspects of a feature together
- Ease the burden of writing a new verb and decrease the chances of bugs
  - Automatic syntactic checks
- Backward compatibility
  - Change only libibverbs commands layer
- Efficient
  - Perfect hash dispatching

OOP based approach



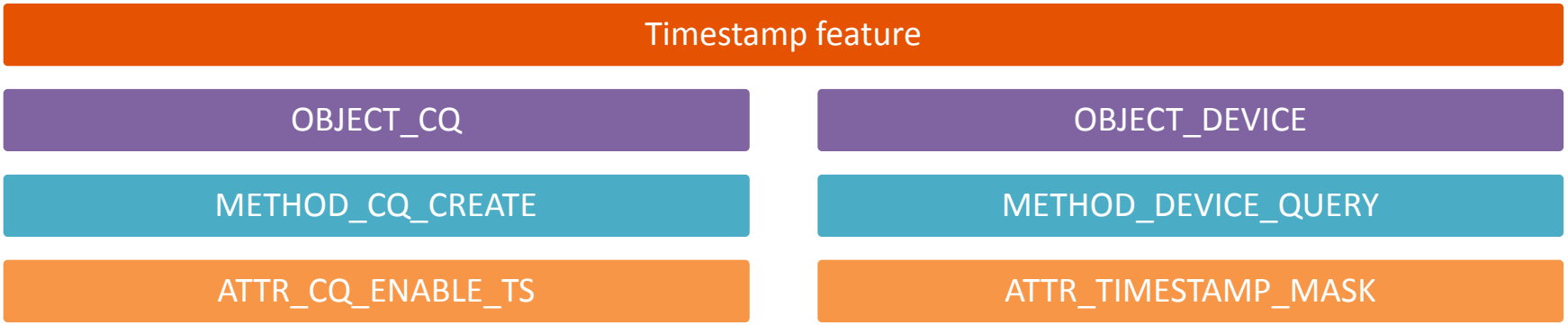
Parsing Trees

- A parsing tree contains a set of objects (defined in previous slide).
- Each feature-set is represented by a parsing tree
  - A feature-set may contain a single feature
- The common feature-set is represented by feature a parsing tree too.
- Objects and methods could exist in few parsing trees
  - Semantically wise, all these objects are conceptually merged
- Driver specific feature-sets are represented by parsing trees.

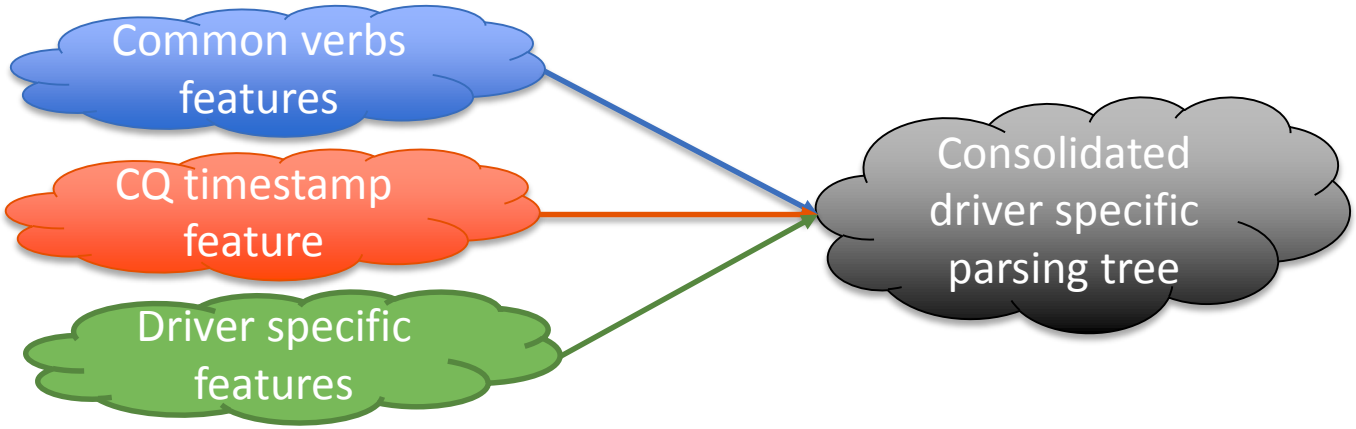


Feature Hierarchy and Merge

- Each feature is represented by a “parsing tree”.



- Driver indicates which features are supported and merge them



Contains only features supported by the device and driver. In the future, this could be queried by the user-space!

Zero conflicts in merge

- Every object, method and attribute is given an **ID**.
  - ID is 16bit, existing for all entities (objects, methods and attributes).
  - IDs are unique in their containing object (i.e. methods in OBJECT_CQ can't collide, but CREATE_CQ and CREATE_QP can have the same ID).
- User-space request is given in a ID-Length-Pointer format. The request is parsed according to these IDs.

Namespace:
0 : Common
1 : Driver Specific
2 – 15 : Reserved

ID in namespace

Domain Specific Language - Object structure

- Parsing-tree is a set of objects:

```
DECLARE_UVERBS_OBJECT_TREE(uverbs_default_objects          /* Name */,
                           &uverbs_object_device          /* Object 1*/,
                           &uverbs_object_pd              /* Object 2 */,
                           &uverbs_object_comp_channel    /* Object 3 */,
                           &uverbs_object_cq              /* Object 4 */,
                           &uverbs_object_qp              /* Object 5 */);
```

- Objects have c'tor, d'tor and other methods

```
DECLARE_UVERBS_OBJECT(uverbs_object_cq /* Name */,
                      UVERBS_OBJECT_CQ /* ID */,
                      &UVERBS_TYPE_ALLOC_IDR_SZ(sizeof(struct ib_ucq_object), 0,
                                                uverbs_free_cq) /* Type info */,
                      &uverbs_method_cq_create /* method 1 */,
                      &uverbs_method_cq_destroy /* method 2 */);
```

Domain specific language – Methods and Attributes

- Methods contain a list of attributes

IDR object, created automatically

```
static DECLARE_UVERBS_METHOD(
        uverbs_method_cq_create /* Name */,
         UVERBS_CQ_CREATE /* ID */,
         uverbs_create_cq_handler /* handler */,
        &UVERBS_ATTR_IDR(CREATE_CQ_HANDLE, UVERBS_OBJECT_CQ, UVERBS_ACCESS_NEW,
                         UA_FLAGS(UVERBS_ATTR_SPEC_F_MANDATORY)),
        &UVERBS_ATTR_PTR_IN(CREATE_CQ_CQE, u32,
                            UA_FLAGS(UVERBS_ATTR_SPEC_F_MANDATORY)),
       &UVERBS_ATTR_FD(CREATE_CQ_COMP_CHANNEL, UVERBS_OBJECT_COMP_CHANNEL,
                       UVERBS_ACCESS_READ),
       &UVERBS_ATTR_PTR_OUT(CREATE_CQ_RESP_CQE, u32,
                            UA_FLAGS(UVERBS_ATTR_SPEC_F_MANDATORY)),
       …
       &uverbs_uhw_compat_in, &uverbs_uhw_compat_out);
```

Driver specific legacy attrs

Framework provides:

- Creating, locking, destroying and mapping IDR/FDs based uobjects
- Automatic size validation for PTR_IN/PTR_OUT
- Automatic validation of mandatory attributes (fail if not exists)

Verbs handler interface


```
static int uverbs_create_cq_handler(struct ib_device *ib_dev,
                                    struct ib_uverbs_file *file,
                                    struct uverbs_attr_bundle *attrs)
```

- Gets “uverbs_attr_bundle” sturct.
- Attributes could be extracted by:
  - `uverbs_attr_get(attrs_bundle, CREATE_CQ_COMP_CHANNEL);` OBJECT
  - `uverbs_copy_from(&dest, attrs_bundle, CREATE_CQ_COMP_VECTOR);` IN_PTR
  - `uverbs_copy_to(attrs_bundle, CREATE_CQ_RESP_CQE, &source);` OUT_PTR

Going forward

- Try the patches and report bugs!
- Gradually start implementing all existing verbs using the new infrastructure. Discuss every verb.
- New driver specific attributes could be passed using the new infrastructure (instead of the legacy ib_udata blob).
  - Replace ib_udata with attribute bundle in all drivers.
- Implement a new query system based on the new driver specific parsing tree.

Discussion Topics

- Implementing user-space API
  - Mix common and driver-specific features
  - Standard ibv types vs driver-specific types
  - Granular API
    - E.g., driver_modify_rc_qp_init_rtr
- Minimal set to enable by default
  - Enough to test traffic
  - E.g., ibv_rc_pingpong
- ABI granularity
  - Do we break up existing calls in kernel?
  - E.g., modify_rc_qp_init_rtr
- New query system
  - Reading the parse-tree
  - Query semantics
- RDMA-CM
  - Using the same infrastructure

BoF

Thank You